In [463]:
```python
# import 'Pandas'
import pandas as pd

# import 'Numpy'
import numpy as np

# import subpackage of Matplotlib
import matplotlib.pyplot as plt

# import 'Seaborn'
import seaborn as sns

# to suppress warnings
from warnings import filterwarnings
filterwarnings('ignore')

# import train-test split
from sklearn.model_selection import train_test_split

# import various functions from statsmodels
import statsmodels
import statsmodels.api as sm

# import 'stats'
from scipy import stats

# 'metrics' from sklearn is used for evaluating the model performance
from sklearn.metrics import mean_squared_error,mean_absolute_error

# import function to perform linear regression
from sklearn.linear_model import LinearRegression,SGDRegressor,Ridge,Lasso,ElasticNet,LogisticRegression

# import StandardScaler to perform scaling
from sklearn.preprocessing import StandardScaler

# import function to perform GridSearchCV
from sklearn.model_selection import GridSearchCV

from mlxtend.feature_selection import SequentialFeatureSelector as sfs

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import cohen_kappa_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import StackingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, ConfusionMatrixDisplay, precision_score,
```

In [464]:
```python
df = pd.read_csv('ILPD.csv')
```

In [465]:
```
df
```

Out[465]:

	Age	Gender	TB	DB	Alkphos	Sgpt	Sgot	TP	ALB	A/G	Selector
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.90	1
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
2	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
3	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
4	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1
...	...	...	...	...	...	...	...	...	...	...	...
578	60	Male	0.5	0.1	500	20	34	5.9	1.6	0.37	2
579	40	Male	0.6	0.1	98	35	31	6.0	3.2	1.10	1
580	52	Male	0.8	0.2	245	48	49	6.4	3.2	1.00	1
581	31	Male	1.3	0.5	184	29	32	6.8	3.4	1.00	1
582	38	Male	1.0	0.3	216	21	24	7.3	4.4	1.50	2

583 rows × 11 columns

In [466]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Age       583 non-null    int64  
 1   Gender    583 non-null    object 
 2   TB        583 non-null    float64
 3   DB        583 non-null    float64
 4   Alkphos   583 non-null    int64  
 5   Sgpt      583 non-null    int64  
 6   Sgot      583 non-null    int64  
 7   TP        583 non-null    float64
 8   ALB       583 non-null    float64
 9   A/G       579 non-null    float64
 10  Selector  583 non-null    int64  
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

In [467]:
```
df['Gender'].unique()
```

Out[467]:
```
array(['Female', 'Male'], dtype=object)
```

In [468]:
```
df['Gender']=df['Gender'].map({'Male':0,'Female':1})
```

In [469]:
```
df['Selector'].unique()
```

Out[469]:
```
array([1, 2], dtype=int64)
```

In [470]:
```
df['Selector']=df['Selector'].map({1:0,2:1})
```

In [471]:
```
df.shape
```

Out[471]:
```
(583, 11)
```

In [472]:
```
df.isnull().sum()
```

Out[472]:
```
Age         0
Gender      0
TB          0
DB          0
Alkphos     0
Sgpt        0
Sgot        0
TP          0
ALB         0
A/G         4
Selector    0
dtype: int64
```

In [473]:
```
df['A/G'].fillna(df['A/G'].mean(),inplace=True)
```

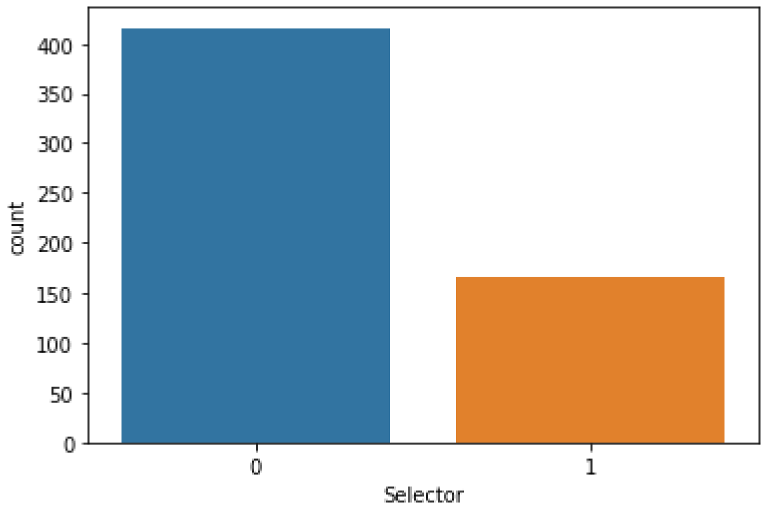
In [474]:
```
df_clm = df.columns
```

In [475]:
```
df_clm
```

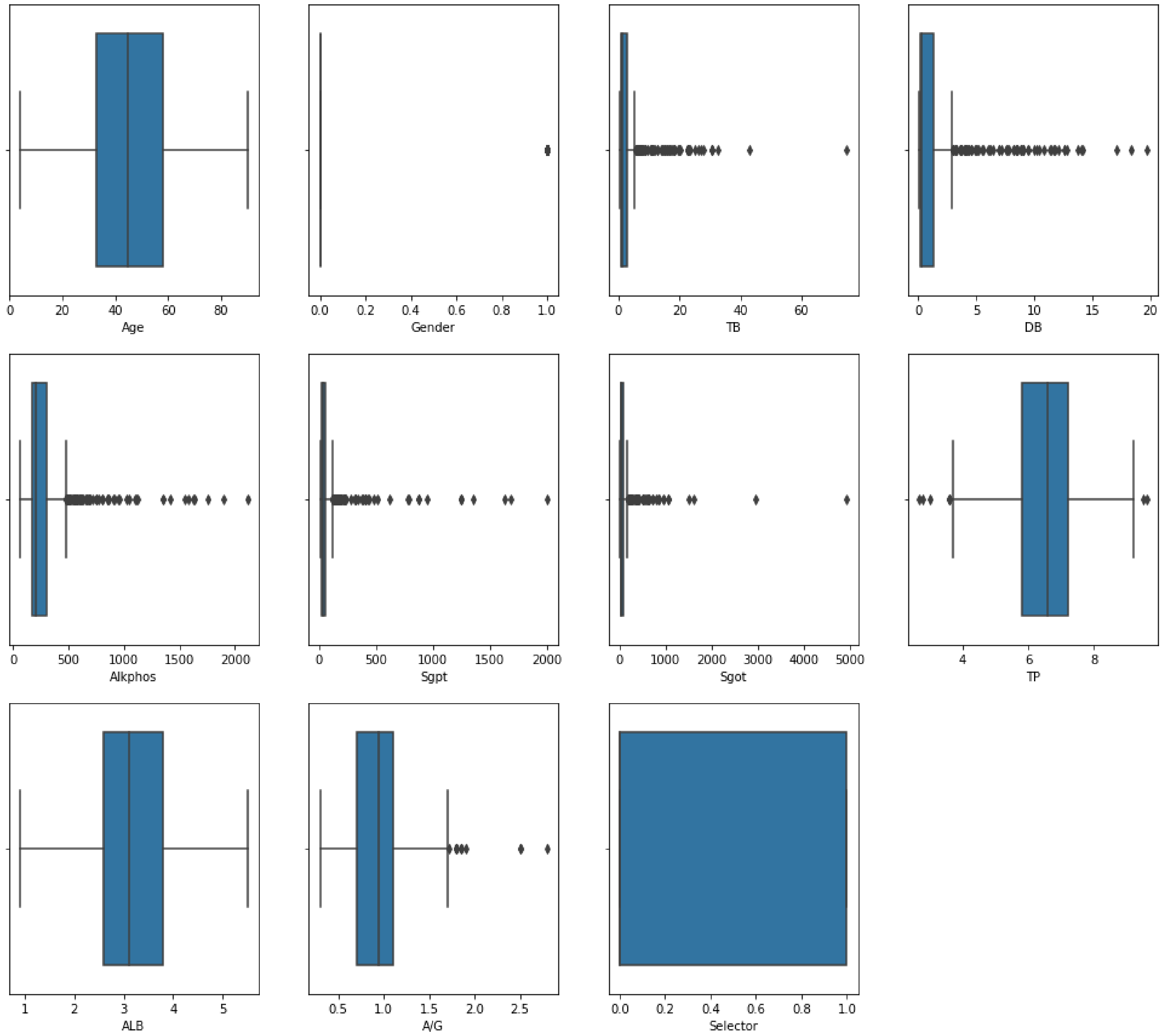
Out[475]:
```
Index(['Age', 'Gender', 'TB', 'DB', 'Alkphos', 'Sgpt', 'Sgot', 'TP', 'ALB',
       'A/G', 'Selector'],
      dtype='object')
```

In [476]:
```
sns.countplot(df.Selector)
```

Out[476]:
```
<AxesSubplot:xlabel='Selector', ylabel='count'>
```

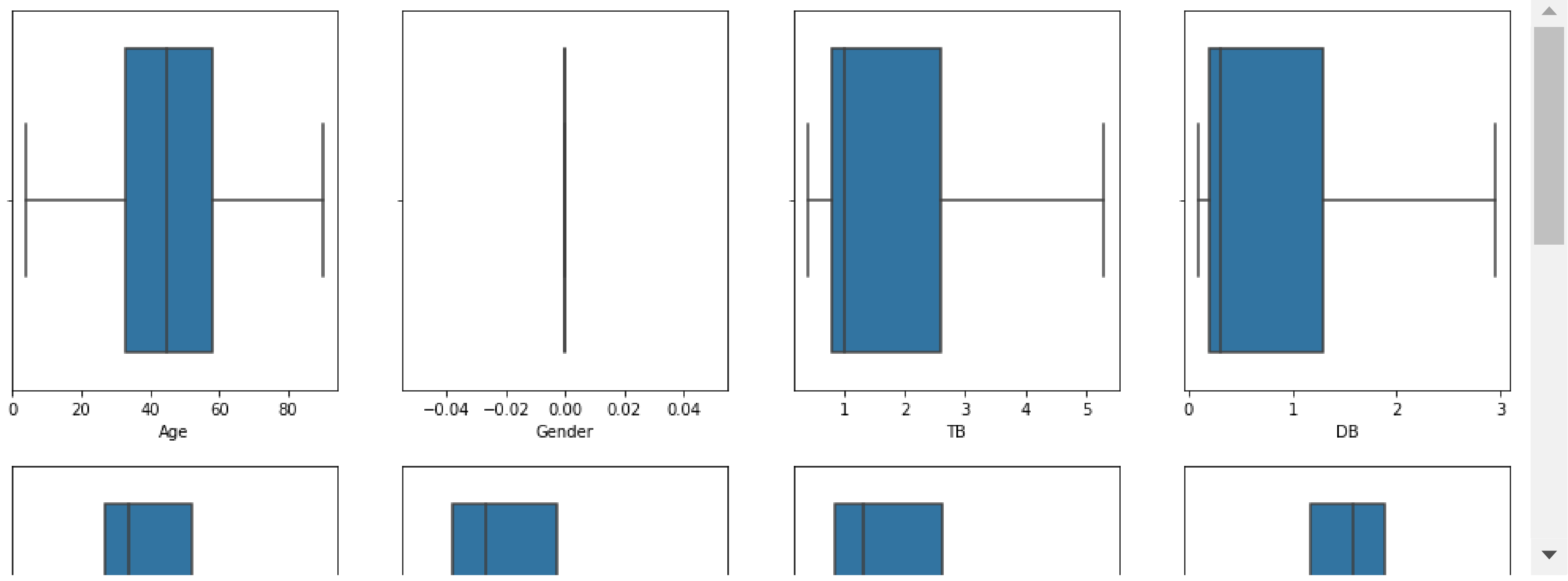


In [477]:
```
t=1
plt.figure(figsize = (17,15))
for i in df_clm:
    plt.subplot(3,4,t)
    sns.boxplot(df[i])
    t+=1
plt.show()
```



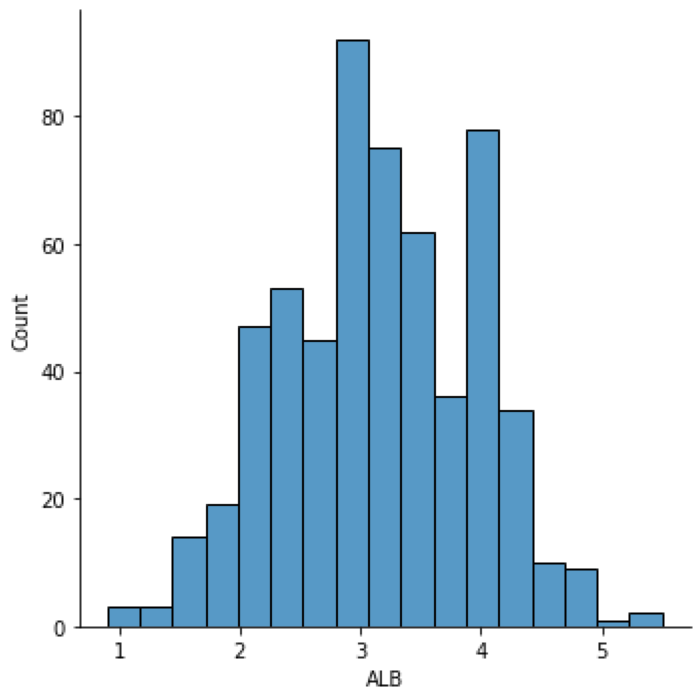
In [478]:
```
for i in df_clm:
    q1,q3 = np.quantile(df[i],[0.25,0.75])
    iqr = q3 - q1
    ub = q3 + (1.5 * iqr)
    lb = q1 - (1.5 * iqr)
    df[i] = np.where(df[i] > ub, ub, df[i])
    df[i] = np.where(df[i] < lb, lb, df[i])
```

In [479]:
```
t=1
plt.figure(figsize = (17,15))
for i in df_clm:
    plt.subplot(3,4,t)
    sns.boxplot(df[i])
    t+=1
plt.show()
```



In [480]:
```
sns.displot(df['ALB'])
```

Out[480]:
```
<seaborn.axisgrid.FacetGrid at 0x28014d55450>
```



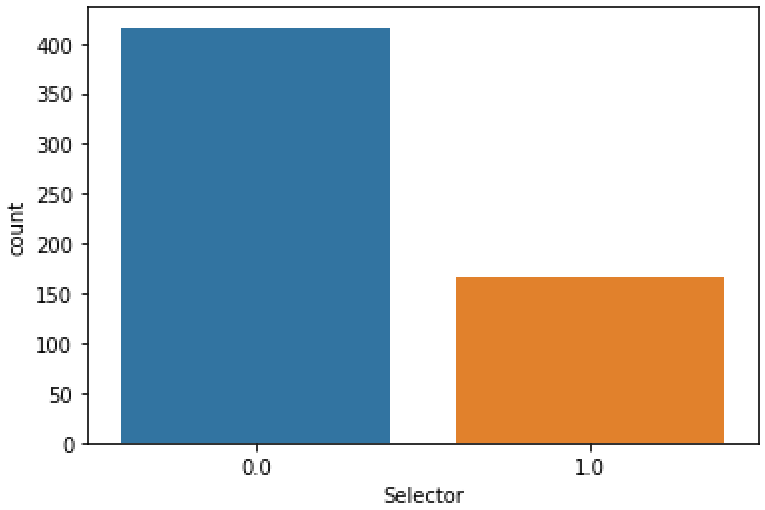
In [481]:
```
# t=1
# plt.figure(figsize = (20,15))
# for i in df_clm:
#     plt.subplot(2,6,t)
#     sns.displot(df[i])
#     t+=1
# plt.show()
```

In [482]:
```
df.skew()
```

Out[482]:
```
Age        -0.029385
Gender      0.000000
TB          1.218379
DB          1.250245
Alkphos     1.036510
Sgpt        1.088208
Sgot        1.188029
TP         -0.203910
ALB        -0.043685
A/G         0.353728
Selector    0.947140
dtype: float64
```

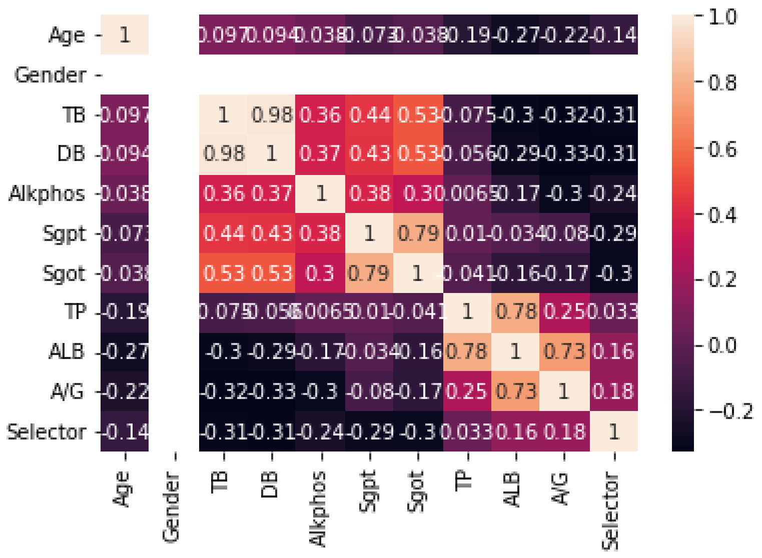
In [483]:
```
sns.countplot(df.Selector)
```

Out[483]: <AxesSubplot:xlabel='Selector', ylabel='count'>



In [484]:
```
sns.heatmap(df.corr(), annot = True)
```

Out[484]: <AxesSubplot:>



In [485]:
```
df = df.drop(['Gender'],axis =1)
```

In [486]:
```
X = df.drop(['Selector'],axis=1)
y = df.Selector
```

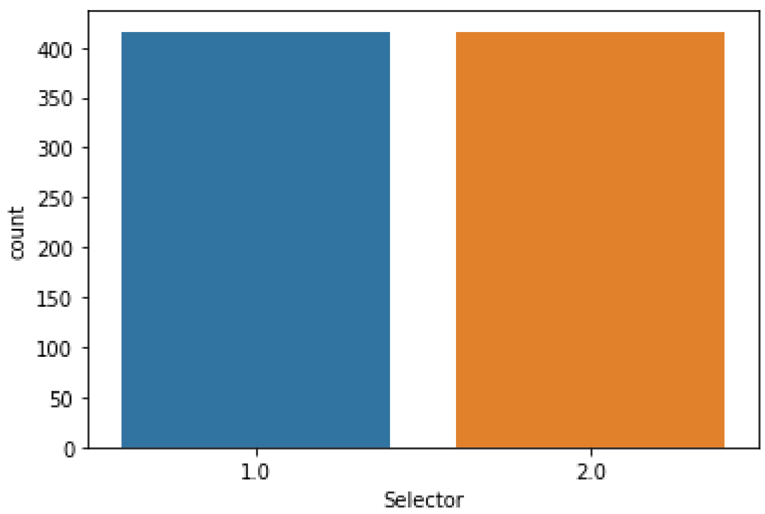
In [487]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=10)
```

If the data is imbalance , use SMOTE Functn

In [488]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X, y = smote.fit_resample(X, y)

df = pd.concat([pd.DataFrame(X), pd.DataFrame(y)], axis=1)
```

In [489]:
```
sns.countplot(df1['Selector'])
```

Out[489]: <AxesSubplot:xlabel='Selector', ylabel='count'>



In [490]:
```
# df.astype(int)
```

In [491]:
```
def get_test_report(model, test_data):
    test_pred = model.predict(test_data)
    return(classification_report(y_test, test_pred))
```

In [492]:
```
def get_train_report(model, train_data):
    train_pred = model.predict(train_data)
    return(classification_report(y_train, train_pred))
```

In [493]:
```
def plot_confusion_matrix(model, test_data):
    y_pred = model.predict(test_data)
    cm = confusion_matrix(y_test, y_pred)
    conf_matrix = pd.DataFrame(data = cm,columns = ['Predicted:0','Predicted:1'], index = ['Actual:0','Actual:1'])
    sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = ListedColormap(['lightskyblue']), cbar = False,
                linewidths = 0.1, annot_kws = {'size':25})
    plt.xticks(fontsize = 20)
    plt.yticks(fontsize = 20)
    plt.show()
```

In [494]:
```
def plot_roc(model, test_data):
    y_pred_prob = model.predict_proba(test_data)[:,1]
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    plt.plot(fpr, tpr)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.plot([0, 1], [0, 1],'r--')
    plt.title('ROC curve for Cancer Prediction Classifier', fontsize = 15)
    plt.xlabel('False positive rate (1-Specificity)', fontsize = 15)
    plt.ylabel('True positive rate (Sensitivity)', fontsize = 15)
    plt.text(x = 0.02, y = 0.9, s = ('AUC Score:',round(roc_auc_score(y_test, y_pred_prob),4)))
    plt.grid(True)
```

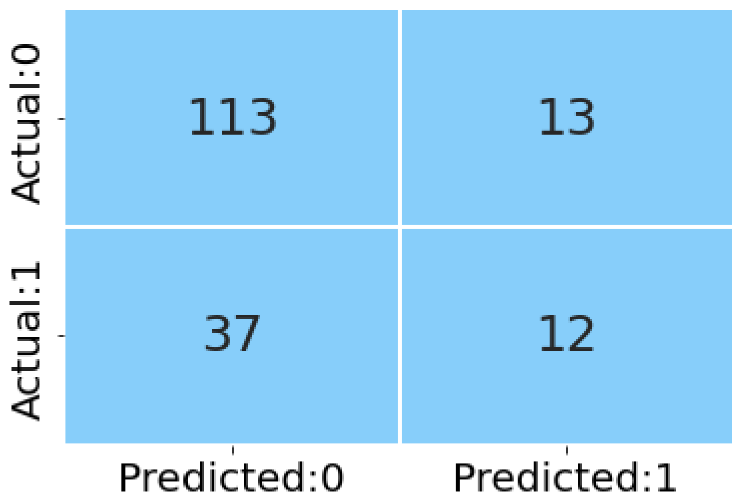
In [ ]:

In [495]:
```
Base_model=LogisticRegression()
Base_model.fit(X_train,y_train)
```

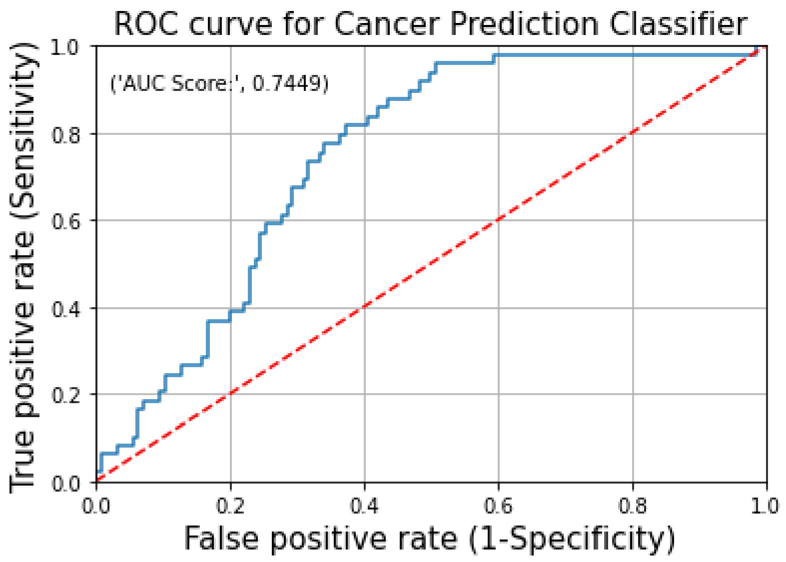
Out[495]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [496]:
```
plot_confusion_matrix(Base_model, test_data = X_test)
```



In [497]:
```
plot_roc(Base_model, test_data = X_test)
```



In [498]:
```python
test_report = get_test_report(Base_model, test_data = X_test)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.75      0.90      0.82       126
         1.0       0.48      0.24      0.32        49

    accuracy                           0.71       175
   macro avg       0.62      0.57      0.57       175
weighted avg       0.68      0.71      0.68       175
```

In [499]:
```python
print('Classification Report for train set: \n', get_train_report(Base_model, train_data = X_train))
```

```
Classification Report for train set:
               precision    recall  f1-score   support

         0.0       0.76      0.92      0.83       290
         1.0       0.59      0.30      0.40       118

    accuracy                           0.74       408
   macro avg       0.68      0.61      0.61       408
weighted avg       0.71      0.74      0.71       408
```

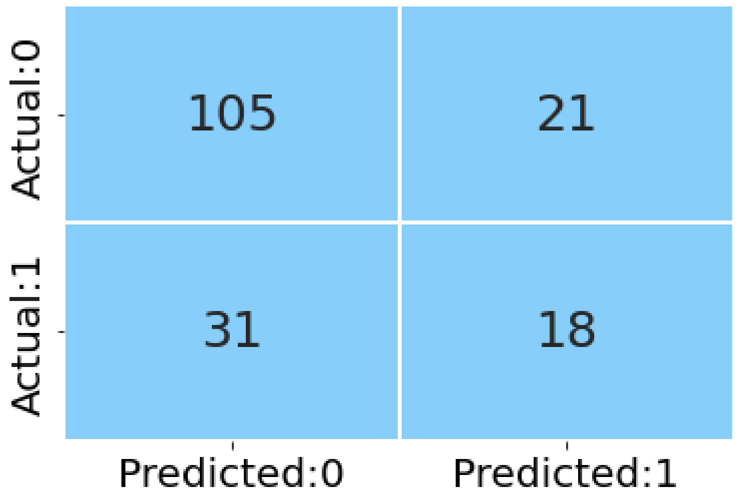
In [500]:
```python
print('Classification Report for test set: \n', get_test_report(Base_model, test_data = X_test))
```

```
Classification Report for test set:
               precision    recall  f1-score   support

         0.0       0.75      0.90      0.82       126
         1.0       0.48      0.24      0.32        49

    accuracy                           0.71       175
   macro avg       0.62      0.57      0.57       175
weighted avg       0.68      0.71      0.68       175
```

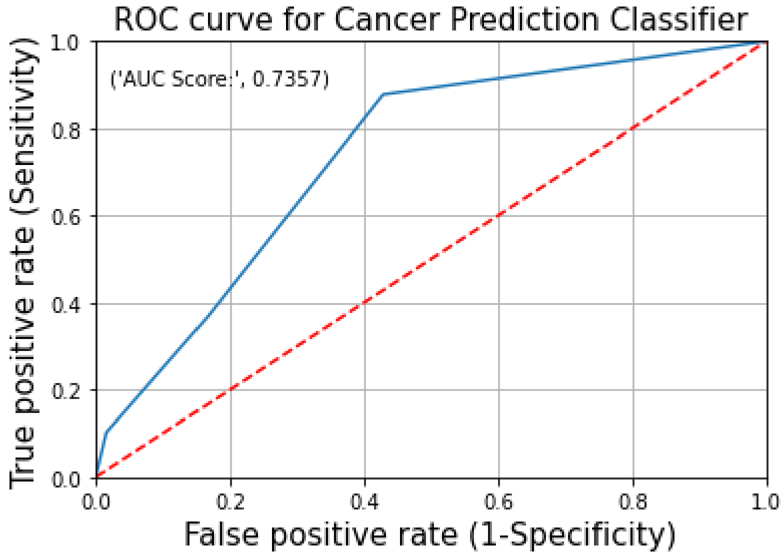
In [ ]:
```python

```

In [501]:
```python
knn_classification = KNeighborsClassifier(n_neighbors = 3)
knn_model = knn_classification.fit(X_train, y_train)
```

In [502]:
```python
plot_confusion_matrix(knn_model, test_data = X_test)
```



In [503]:
```python
plot_roc(knn_model, test_data = X_test)
```



KNN Classification GridSearchCV

In [504]:
```
tuned_paramaters = {'n_neighbors': np.arange(1, 25, 2),
                    'metric': ['hamming','euclidean','manhattan','Chebyshev']}

# instantiate the 'KNeighborsClassifier'
knn_classification = KNeighborsClassifier()
knn_grid = GridSearchCV(estimator = knn_classification,
                        param_grid = tuned_paramaters,
                        cv = 5,
                        scoring = 'accuracy')
knn_grid.fit(X_train, y_train)
print('Best parameters for KNN Classifier: ', knn_grid.best_params_, '\n')
```

```
Best parameters for KNN Classifier:  {'metric': 'euclidean', 'n_neighbors': 23}
```

In [505]:
```
knn_classification = KNeighborsClassifier(n_neighbors = 1,metric= 'manhattan')
knn_model_hp = knn_classification.fit(X_train, y_train)
```

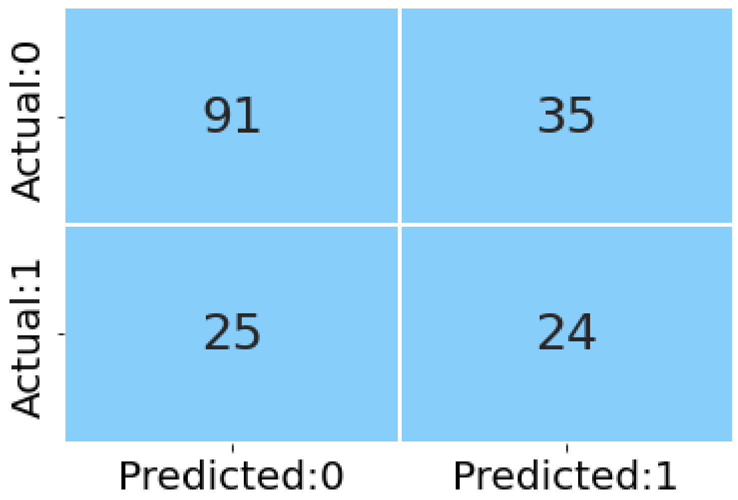
In [506]:
```
train_report = get_train_report(knn_model_hp, train_data = X_train)
print(train_report)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       290
         1.0       1.00      1.00      1.00       118

    accuracy                           1.00       408
   macro avg       1.00      1.00      1.00       408
weighted avg       1.00      1.00      1.00       408
```

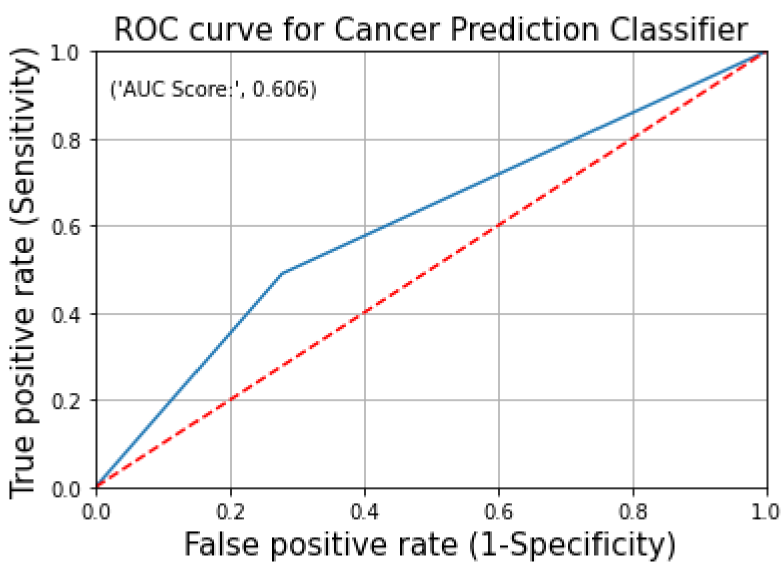
In [507]:
```
test_report = get_test_report(knn_model_hp, test_data = X_test)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.78      0.72      0.75       126
         1.0       0.41      0.49      0.44        49

    accuracy                           0.66       175
   macro avg       0.60      0.61      0.60       175
weighted avg       0.68      0.66      0.67       175
```

In [508]:
```
plot_confusion_matrix(knn_model_hp, test_data = X_test)
```



In [509]:
```
plot_roc(knn_model_hp, test_data = X_test)
```

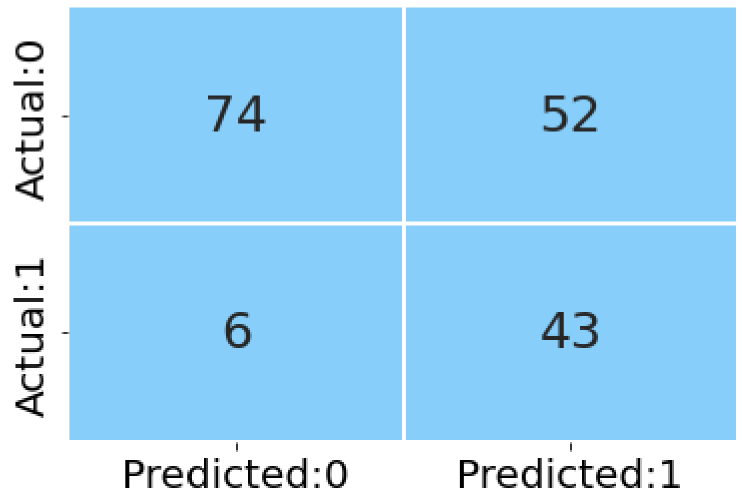


In [ ]:

GaussianNB

In [510]:
```
gnb = GaussianNB()
gnb_model = gnb.fit(X_train, y_train)
```

In [511]:
```
plot_confusion_matrix(gnb_model, test_data=X_test)
```



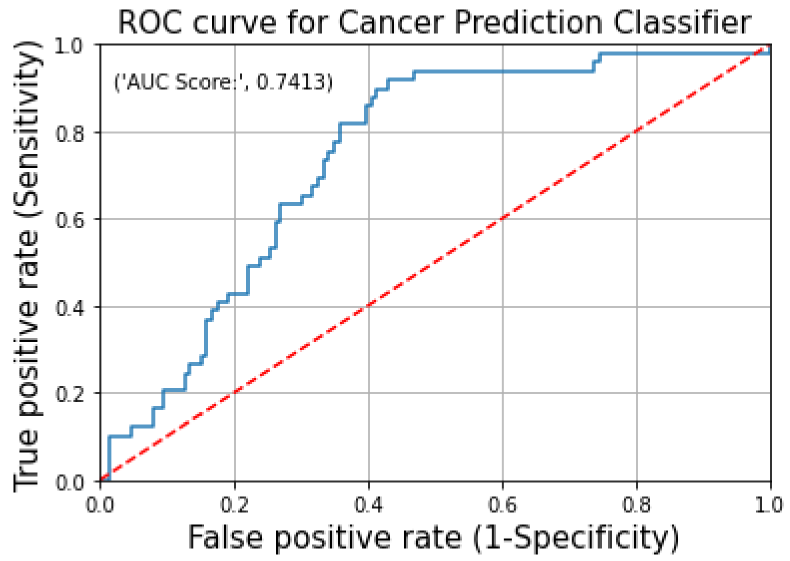
In [512]:
```
train_report = get_train_report(gnb_model, train_data=X_train)
print(train_report)
```

```
              precision    recall  f1-score   support

         0.0       0.89      0.53      0.67       290
         1.0       0.42      0.84      0.56       118

    accuracy                           0.62       408
   macro avg       0.66      0.69      0.61       408
weighted avg       0.75      0.62      0.64       408
```

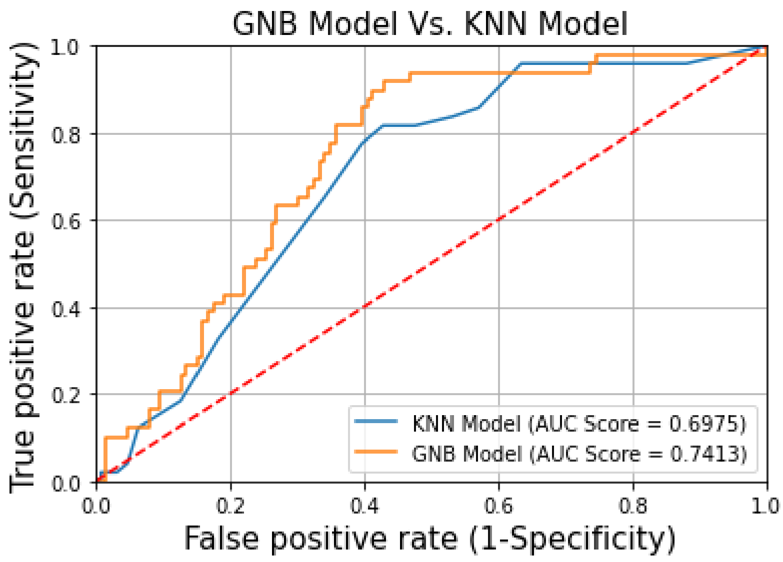
In [513]:
```
test_report = get_test_report(gnb_model, test_data=X_test)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.59      0.72       126
         1.0       0.45      0.88      0.60        49

    accuracy                           0.67       175
   macro avg       0.69      0.73      0.66       175
weighted avg       0.79      0.67      0.68       175
```

In [514]:
```
plot_roc(gnb_model, test_data=X_test)
```



In [515]:
```
y_pred_prob_knn = knn_grid.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_knn)
auc_score_knn = roc_auc_score(y_test, y_pred_prob_knn)
plt.plot(fpr, tpr, label='KNN Model (AUC Score = %0.4f)' % auc_score_knn)
y_pred_prob_gnb = gnb_model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_gnb)
auc_score_gnb = roc_auc_score(y_test, y_pred_prob_gnb)
plt.plot(fpr, tpr, label='GNB Model (AUC Score = %0.4f)' % auc_score_gnb)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.plot([0, 1], [0, 1],'r--')
plt.title('GNB Model Vs. KNN Model', fontsize = 15)
plt.xlabel('False positive rate (1-Specificity)', fontsize = 15)
plt.ylabel('True positive rate (Sensitivity)', fontsize = 15)
plt.legend(loc = 'lower right')
plt.grid(True)
```



Decision Tree Classification

In [516]:
```
decision_tree_classification = DecisionTreeClassifier(criterion = 'entropy', random_state = 10)
decision_tree = decision_tree_classification.fit(X_train, y_train)
```

In [517]:
```
train_report = get_train_report(decision_tree, train_data=X_train)
print(train_report)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       290
         1.0       1.00      1.00      1.00       118

    accuracy                           1.00       408
   macro avg       1.00      1.00      1.00       408
weighted avg       1.00      1.00      1.00       408
```

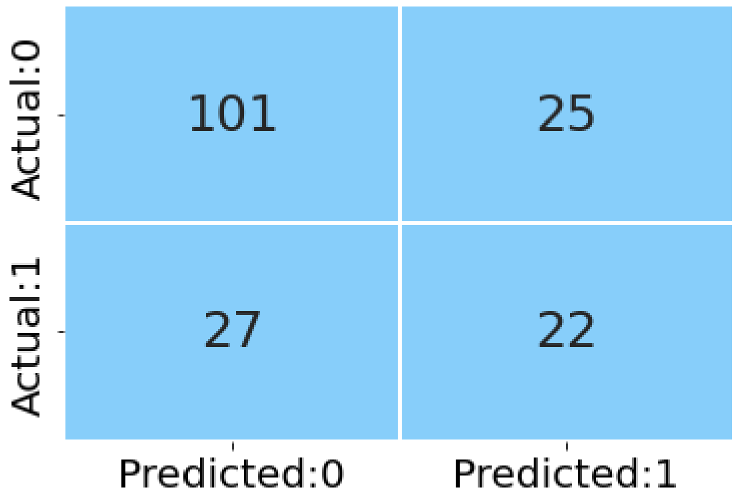
In [518]:
```
test_report = get_test_report(decision_tree, test_data=X_test)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.79      0.80      0.80       126
         1.0       0.47      0.45      0.46        49

    accuracy                           0.70       175
   macro avg       0.63      0.63      0.63       175
weighted avg       0.70      0.70      0.70       175
```

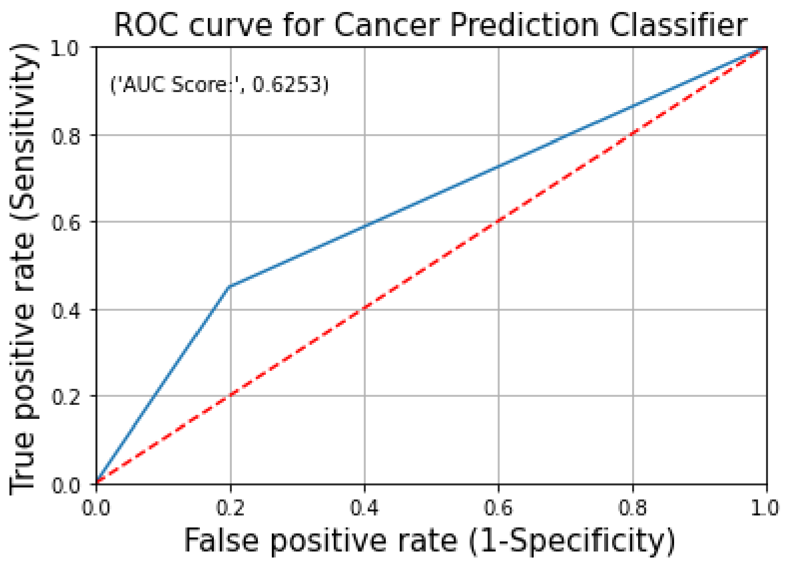
Interpretation: From the above output, we can see that there is a difference between the train and test accuracy; thus, we can conclude that the decision tree is over-fitted on the train data.

if we tune the hyperparameters in the decision tree, it helps to avoid the over-fitting of the tree.

In [519]:
```
plot_confusion_matrix(decision_tree, test_data = X_test)
```



In [520]:
```
plot_roc(decision_tree, test_data=X_test)
```



In [521]:
```
dt_model = DecisionTreeClassifier(criterion = 'gini',
                                  max_depth = 5,
                                  min_samples_split = 4,
                                  max_leaf_nodes = 6,
                                  random_state = 10)
decision_tree = dt_model.fit(X_train, y_train)
train_report = get_train_report(decision_tree, train_data=X_train)
print('Train data:\n', train_report)
test_report = get_test_report(decision_tree, test_data=X_test)
print('Test data:\n', test_report)
```

```
Train data:
               precision    recall  f1-score   support

         0.0       0.85      0.79      0.82       290
         1.0       0.56      0.65      0.60       118

    accuracy                           0.75       408
   macro avg       0.70      0.72      0.71       408
weighted avg       0.76      0.75      0.76       408

Test data:
               precision    recall  f1-score   support

         0.0       0.75      0.71      0.73       126
         1.0       0.35      0.41      0.38        49

    accuracy                           0.62       175
   macro avg       0.55      0.56      0.55       175
weighted avg       0.64      0.62      0.63       175
```

Interpretation: From the above output, we can see that there is slight significant difference between the train and test accuracy; thus, we can conclude that the decision tree is less over-fiited after specifying some of the hyperparameters.

Decision Tree Classification GridSearchCV

In [522]:
```
# tuned_paramaters = [{'criterion': ['entropy', 'gini'],
#                        'max_depth': range(2, 10),
#                        'max_features': ["sqrt", "log2"],
#                        'min_samples_split': range(2,10),
#                        'min_samples_leaf': range(1,10),
#                        'max_leaf_nodes': range(1, 10)}]
# decision_tree_classification = DecisionTreeClassifier(random_state = 10)
# tree_grid = GridSearchCV(estimator = decision_tree_classification,
#                           param_grid = tuned_paramaters,
#                           cv = 5)
# tree_grid_model = tree_grid.fit(X_train, y_train)
# print('Best parameters for decision tree classifier: ', tree_grid_model.best_params_, '\n')
```

In [523]:
```
# dt_model = DecisionTreeClassifier(criterion = tree_grid_model.best_params_.get('criterion'),
#                                    max_depth = tree_grid_model.best_params_.get('max_depth'),
#                                    max_features = tree_grid_model.best_params_.get('max_features'),
#                                    max_leaf_nodes = tree_grid_model.best_params_.get('max_leaf_nodes'),
#                                    min_samples_leaf = tree_grid_model.best_params_.get('min_samples_leaf'),
#                                    min_samples_split = tree_grid_model.best_params_.get('min_samples_split'),
#                                    random_state = 10)

# # use fit() to fit the model on the train set
# dt_model = dt_model.fit(X_train, y_train)
```

In [524]:
```
# print('Classification Report for train set: \n', get_train_report(dt_model, train_data = X_train))
```

In [525]:
```
# print('Classification Report for test set: \n', get_test_report(dt_model, test_data = X_test))
```

Interpretation: From the above output, we can see that there is no significant difference between the train and test accuracy; thus, we can conclude that the decision tree after tuning the hyperparameters avoids the over-fitting of the data.

Random Forest classification

In [526]:
```
rf_classification = RandomForestClassifier(n_estimators = 10, random_state = 10)
rf_model = rf_classification.fit(X_train, y_train)
```

In [527]:
```
train_report = get_train_report(rf_model,train_data = X_train)
print(train_report)
```

```
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99       290
         1.0       1.00      0.94      0.97       118

    accuracy                           0.98       408
   macro avg       0.99      0.97      0.98       408
weighted avg       0.98      0.98      0.98       408
```

In [528]:
```
test_report = get_test_report(rf_model, test_data = X_test)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.76      0.87      0.81       126
         1.0       0.47      0.31      0.37        49

    accuracy                           0.71       175
   macro avg       0.62      0.59      0.59       175
weighted avg       0.68      0.71      0.69       175
```

Random Forest Classification Grid Search CV

In [529]:
```
# tuned_paramaters_rf = [{'criterion': ['entropy', 'gini'],
#                        'n_estimators': [10, 30, 50, 70, 90],
#                        'max_depth': [10, 15, 20],
#                        'max_features': ['sqrt', 'log2'],
#                        'min_samples_split': [2, 5, 8, 11],
#                        'min_samples_leaf': [1, 5, 9],
#                        'max_leaf_nodes': [2, 5, 8, 11]}]

# # instantiate the 'RandomForestClassifier'
# # pass the 'random_state' to obtain the same samples for each time you run the code
# random_forest_classification = RandomForestClassifier(random_state = 10)

# # use GridSearchCV() to find the optimal value of the hyperparameters
# # estimator: pass the random forest classifier model
# # param_grid: pass the list 'tuned_parameters'
# # cv: number of folds in k-fold i.e. here cv = 5
# rf_grid = GridSearchCV(estimator = random_forest_classification,
#                          param_grid = tuned_paramaters_rf,
#                          cv = 5)

# # use fit() to fit the model on the train set
# rf_grid_model = rf_grid.fit(X_train, y_train)

# # get the best parameters
# print('Best parameters for random forest classifier: ', rf_grid_model.best_params_, '\n')
```

In [530]:
```
# rf_model = RandomForestClassifier(criterion = rf_grid_model.best_params_.get('criterion'),
#                                    n_estimators = rf_grid_model.best_params_.get('n_estimators'),
#                                    max_depth = rf_grid_model.best_params_.get('max_depth'),
#                                    max_features = rf_grid_model.best_params_.get('max_features'),
#                                    max_leaf_nodes = rf_grid_model.best_params_.get('max_leaf_nodes'),
#                                    min_samples_leaf = rf_grid_model.best_params_.get('min_samples_leaf'),
#                                    min_samples_split = rf_grid_model.best_params_.get('min_samples_split'),
#                                    random_state = 10)
# rf_model = rf_model.fit(X_train, y_train)
# print('Classification Report for test set:\n', get_test_report(rf_model,test_data = X_test))
```

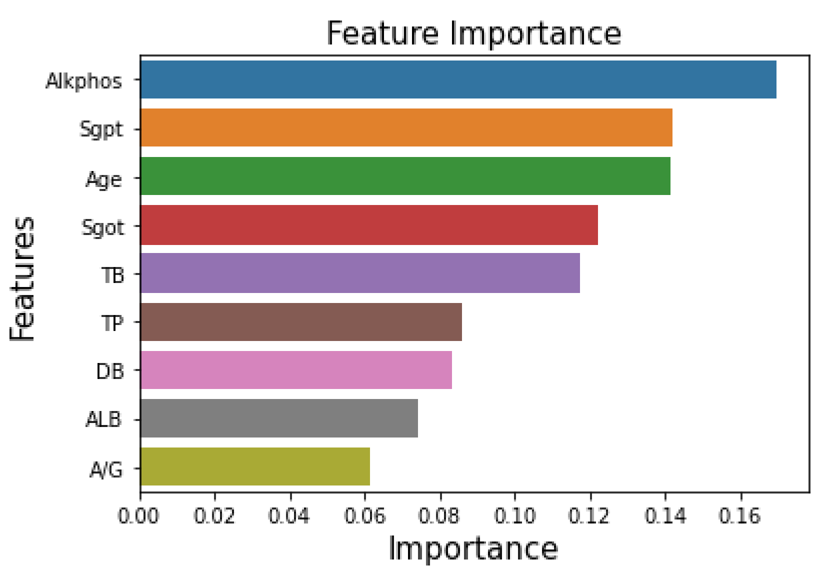
In [531]:
```
# print('Classification Report for Train set:\n', get_train_report(rf_model,train_data = X_train))
```

In [532]:
```
# plot_roc(rf_model,test_data=X_test)
```

In [533]:
```
# plot_confusion_matrix(rf_model, test_data=X_test)
```

Interpretation: The accuracy of the test dataset increased from 0.81 to 0.82 after tuning of the hyperparameters. Also, the sensitivity and specificity of the model are balanced.

In [534]:
```
important_features = pd.DataFrame({'Features': X_train.columns,
                                   'Importance': rf_model.feature_importances_})
important_features = important_features.sort_values('Importance', ascending = False)
sns.barplot(x = 'Importance', y = 'Features', data = important_features)
plt.title('Feature Importance', fontsize = 15)
plt.xlabel('Importance', fontsize = 15)
plt.ylabel('Features', fontsize = 15)
plt.show()
```



In [ ]:

Ada Boost

In [535]:
```
ada_model = AdaBoostClassifier(n_estimators = 40, random_state = 10)
ada_model.fit(X_train, y_train)
```

Out[535]:
```
AdaBoostClassifier(n_estimators=40, random_state=10)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [536]:
```
test_report = get_train_report(ada_model,train_data = X_train)
print(test_report)
```

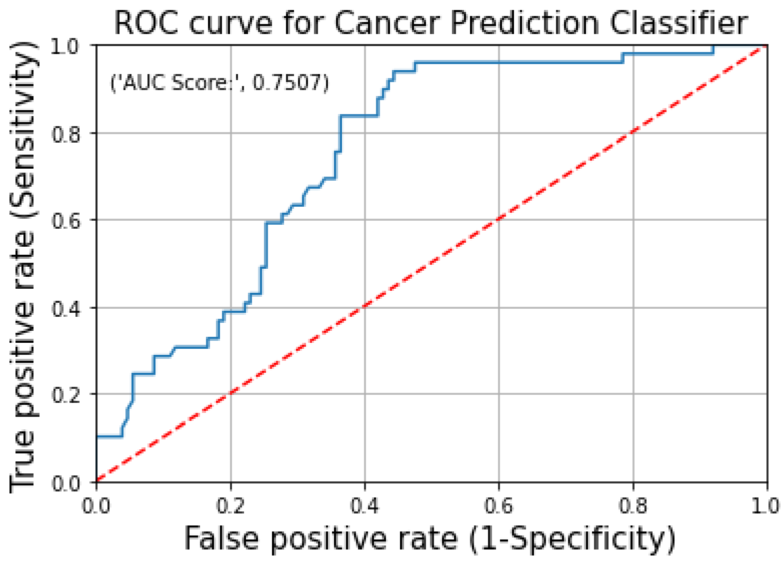
```
              precision    recall  f1-score   support

         0.0       0.85      0.90      0.87       290
         1.0       0.72      0.60      0.65       118

    accuracy                           0.82       408
   macro avg       0.78      0.75      0.76       408
weighted avg       0.81      0.82      0.81       408
```

In [537]:
```
test_report = get_test_report(ada_model,test_data = X_test)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.76      0.82      0.79       126
         1.0       0.42      0.35      0.38        49

    accuracy                           0.69       175
   macro avg       0.59      0.58      0.59       175
weighted avg       0.67      0.69      0.68       175
```

Interpretation: The output shows that the model is 83% accurate.

In [538]:
```
plot_roc(ada_model,test_data=X_test)
```



Gradient Boosting Classification

In [539]:
```
gboost_model = GradientBoostingClassifier(n_estimators = 150, max_depth = 10, random_state = 10)
gboost_model.fit(X_train, y_train)
```

Out[539]:
```
GradientBoostingClassifier(max_depth=10, n_estimators=150, random_state=10)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [540]:
```
test_report = get_train_report(gboost_model,train_data = X_train)
print(test_report)
```

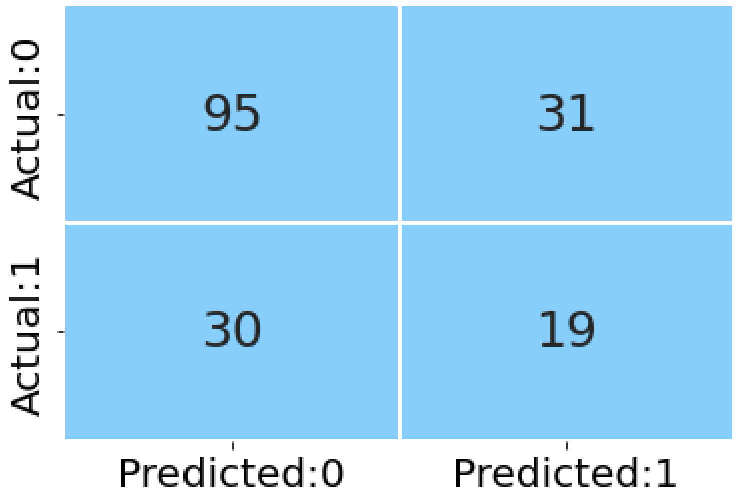
```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       290
         1.0       1.00      1.00      1.00       118

    accuracy                           1.00       408
   macro avg       1.00      1.00      1.00       408
weighted avg       1.00      1.00      1.00       408
```

In [541]:
```
test_report = get_test_report(gboost_model,test_data = X_test)
print(test_report)
```

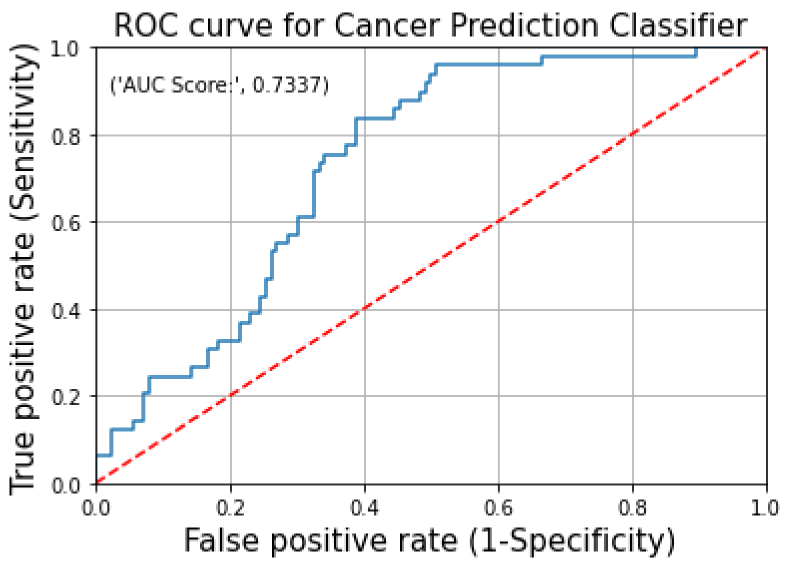
```
              precision    recall  f1-score   support

         0.0       0.76      0.75      0.76       126
         1.0       0.38      0.39      0.38        49

    accuracy                           0.65       175
   macro avg       0.57      0.57      0.57       175
weighted avg       0.65      0.65      0.65       175
```

In [ ]:

In [542]:
```
plot_confusion_matrix(gboost_model, test_data=X_test)
```



In [543]:
```
plot_roc(gboost_model,test_data = X_test)
```



XGB Classification

In [544]:
```
xgb_model = XGBClassifier(max_depth = 10, gamma = 1)
xgb_model.fit(X_train, y_train)
```

Out[544]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=1, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=10, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

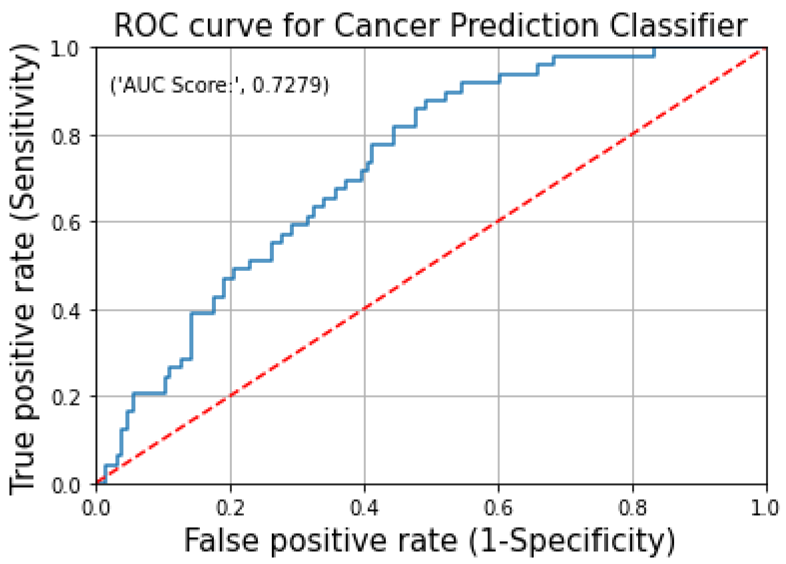
In [545]:
```python
test_report = get_train_report(xgb_model,train_data = X_train)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       290
         1.0       1.00      0.98      0.99       118

    accuracy                           1.00       408
   macro avg       1.00      0.99      0.99       408
weighted avg       1.00      1.00      1.00       408
```

In [546]:
```python
test_report = get_test_report(xgb_model,test_data = X_test)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.78      0.83      0.80       126
         1.0       0.46      0.39      0.42        49

    accuracy                           0.70       175
   macro avg       0.62      0.61      0.61       175
weighted avg       0.69      0.70      0.69       175
```

In [547]:
```python
plot_roc(xgb_model,test_data = X_test)
```



XGB Classification Grid Search CV

In [548]:
```python
tuning_parameters = {'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6],
                     'max_depth': range(3,10),
                     'gamma': [0, 1, 2, 3, 4]}
xgb_model = XGBClassifier()
xgb_grid = GridSearchCV(estimator = xgb_model, param_grid = tuning_parameters, cv = 3, scoring = 'roc_auc')
xgb_grid.fit(X_train, y_train)
print('Best parameters for XGBoost classifier: ', xgb_grid.best_params_, '\n')
```

```
Best parameters for XGBoost classifier:  {'gamma': 4, 'learning_rate': 0.2, 'max_depth': 6}
```

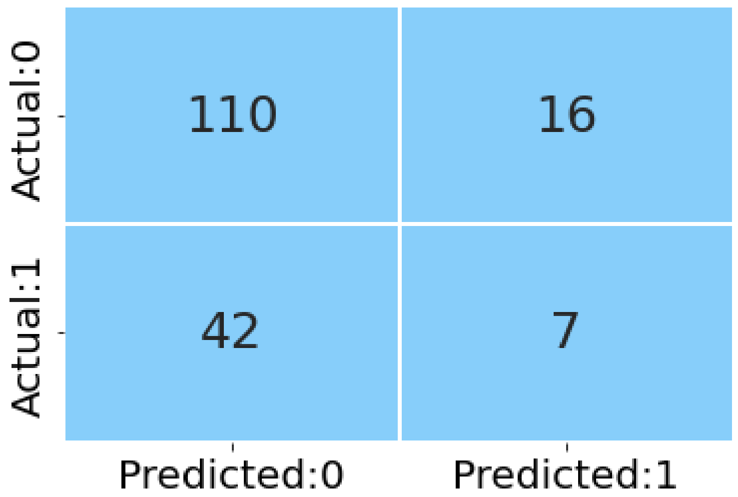
In [549]:
```python
xgb_grid_model = XGBClassifier(learning_rate = xgb_grid.best_params_.get('learning_rate'),
                               max_depth = xgb_grid.best_params_.get('max_depth'),
                              gamma = xgb_grid.best_params_.get('gamma'))
xgb_model = xgb_grid_model.fit(X_train, y_train)
print('Classification Report for test set:\n', get_test_report(xgb_model,test_data = X_test))
```

```
Classification Report for test set:
               precision    recall  f1-score   support

         0.0       0.72      0.87      0.79       126
         1.0       0.30      0.14      0.19        49

    accuracy                           0.67       175
   macro avg       0.51      0.51      0.49       175
weighted avg       0.61      0.67      0.62       175
```

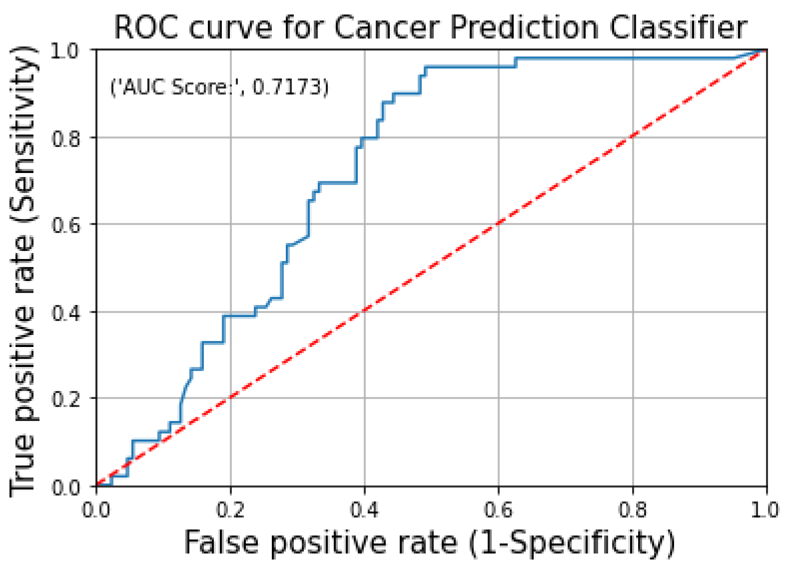
In [550]:
```
print('Classification Report for Train set:\n', get_train_report(xgb_model,train_data = X_train))
```

```
Classification Report for Train set:
               precision    recall  f1-score   support

         0.0       0.84      0.97      0.90       290
         1.0       0.89      0.53      0.67       118

    accuracy                           0.85       408
   macro avg       0.86      0.75      0.78       408
weighted avg       0.85      0.85      0.83       408
```

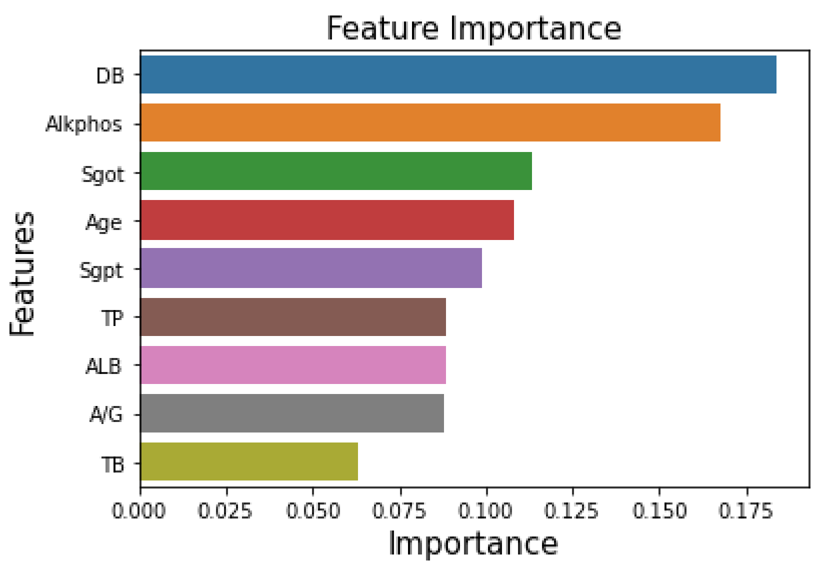
In [551]:
```
plot_confusion_matrix(xgb_grid_model, test_data=X_test)
```



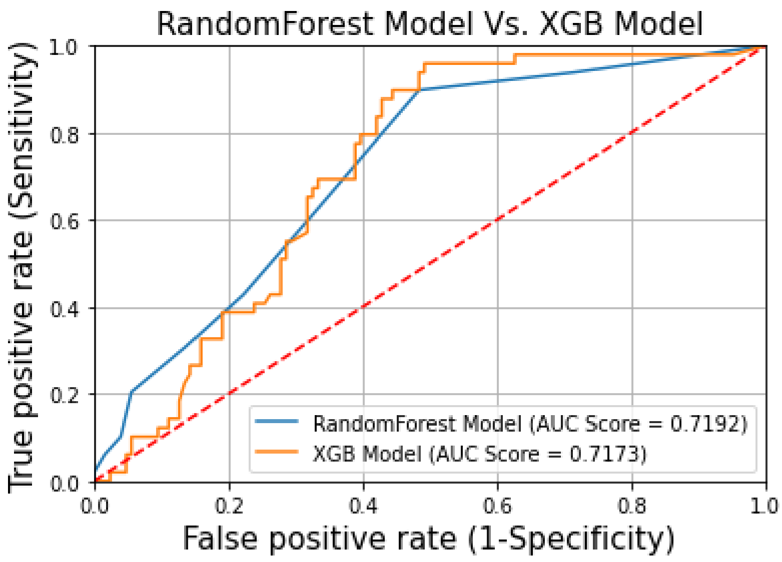
In [552]:
```
plot_roc(xgb_model,test_data = X_test)
```



In [553]:
```
important_features = pd.DataFrame({'Features': X_train.columns,
                                    'Importance': xgb_model.feature_importances_})
important_features = important_features.sort_values('Importance', ascending = False)
sns.barplot(x = 'Importance', y = 'Features', data = important_features)
plt.title('Feature Importance', fontsize = 15)
plt.xlabel('Importance', fontsize = 15)
plt.ylabel('Features', fontsize = 15)
plt.show()
```



In [554]:
```
y_pred_prob_rf = rf_model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_rf)
auc_score_rf = roc_auc_score(y_test, y_pred_prob_rf)
plt.plot(fpr, tpr, label='RandomForest Model (AUC Score = %0.4f)' % auc_score_rf)
y_pred_prob_xgb = xgb_grid_model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_xgb)
auc_score_xgb = roc_auc_score(y_test, y_pred_prob_xgb)
plt.plot(fpr, tpr, label='XGB Model (AUC Score = %0.4f)' % auc_score_xgb)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.plot([0, 1], [0, 1],'r--')
plt.title('RandomForest Model Vs. XGB Model', fontsize = 15)
plt.xlabel('False positive rate (1-Specificity)', fontsize = 15)
plt.ylabel('True positive rate (Sensitivity)', fontsize = 15)
plt.legend(loc = 'lower right')
plt.grid(True)
```



Stacking Classification

In [555]:
```
# consider the various algorithms as base learners
base_learners = [('rf_model', RandomForestClassifier(criterion = 'entropy', max_depth = 10, max_features = 'sqrt',
                                                     max_leaf_nodes = 8, min_samples_leaf = 5, min_samples_split = 2,
                                                     n_estimators = 50, random_state = 10)),
                 ('KNN_model', KNeighborsClassifier(n_neighbors = 17, metric = 'euclidean')),
                 ('NB_model', GaussianNB())]

# initialize stacking classifier
# pass the base learners to the parameter, 'estimators'
# pass the Naive Bayes model as the 'final_estimator'/ meta model
stack_model = StackingClassifier(estimators = base_learners, final_estimator = GaussianNB(),)

# fit the model on train dataset
stack_model.fit(X_train, y_train)
```

Out[555]:
```
StackingClassifier(estimators=[('rf_model',
                                RandomForestClassifier(criterion='entropy',
                                                       max_depth=10,
                                                       max_leaf_nodes=8,
                                                       min_samples_leaf=5,
                                                       n_estimators=50,
                                                       random_state=10)),
                               ('KNN_model',
                                KNeighborsClassifier(metric='euclidean',
                                                     n_neighbors=17)),
                               ('NB_model', GaussianNB())],
                   final_estimator=GaussianNB())
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [556]:
```
test_report = get_train_report(stack_model,train_data = X_train)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.66      0.75       290
         1.0       0.48      0.79      0.60       118

    accuracy                           0.69       408
   macro avg       0.68      0.72      0.68       408
weighted avg       0.77      0.69      0.71       408
```

In [557]:
```
test_report = get_test_report(stack_model,test_data = X_test)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.65      0.75       126
         1.0       0.46      0.78      0.58        49

    accuracy                           0.69       175
   macro avg       0.67      0.71      0.66       175
weighted avg       0.76      0.69      0.70       175
```

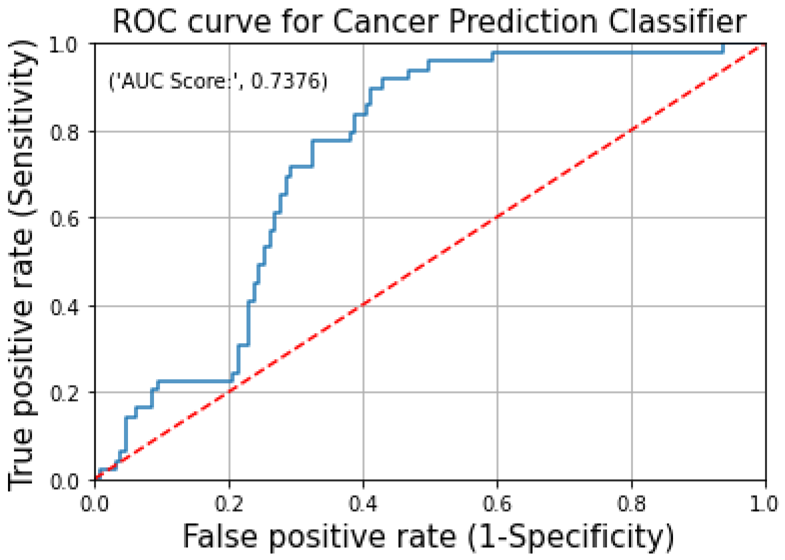
In [574]:
```
crossvalst = cross_val_score(stack_model, X_train, y_train, cv=5)
mean_accuracy = crossvalst.mean()
mean_accuracy
```

Out[574]: 0.6468834688346883

In [575]:
```
crossvalst = cross_val_score(stack_model, X_test, y_test, cv=5)
mean_accuracy = crossvalst.mean()
mean_accuracy
```

Out[575]: 0.6628571428571429

In [558]:
```
plot_roc(stack_model,test_data = X_test)
```



In [ ]:

In [559]:
```
model = LogisticRegression()
crossval = cross_val_score(model, X_train, y_train, cv=5)  # 5-fold cross-validation
```

In [571]:
```
mean_accuracy_train = crossval.mean()
mean_accuracy_train
```

Out[571]: 0.7205058717253839

In [572]:
```
model = LogisticRegression()
crossvalt = cross_val_score(model, X_test, y_test, cv=5)  # 5-fold cross-validation
```

In [573]:
```
mean_accuracy = crossvalt.mean()
mean_accuracy
```

Out[573]: 0.6742857142857144

In [570]:
```
model = LogisticRegression(penalty='l2')  # L2 regularization
model.fit(X_train, y_train)
```

Out[570]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [565]:
```
test_report = get_train_report(model,train_data = X_train)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.76      0.92      0.83       290
         1.0       0.59      0.30      0.40       118

    accuracy                           0.74       408
   macro avg       0.68      0.61      0.61       408
weighted avg       0.71      0.74      0.71       408
```

In [566]:
```
test_report = get_test_report(model,test_data = X_test)
print(test_report)
```

```
              precision    recall  f1-score   support

         0.0       0.75      0.90      0.82       126
         1.0       0.48      0.24      0.32        49

    accuracy                           0.71       175
   macro avg       0.62      0.57      0.57       175
weighted avg       0.68      0.71      0.68       175
```

In [ ]: